03-27-2023

CS051A

INTRO TO COMPUTER SCIENCE WITH TOPICS IN AI

16: More classes

Alexandra Papoutsaki

she/her/hers

Lectures

Zilong Ye

he/him/his

Labs

Lecture 16: More classes

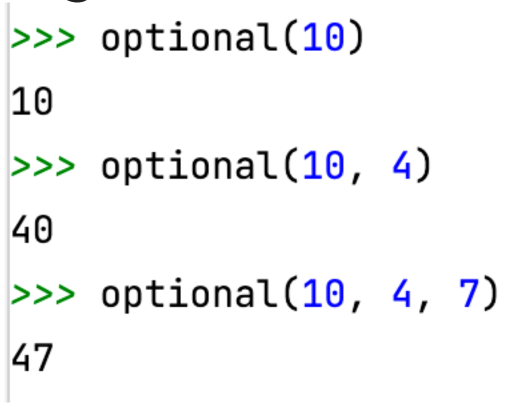
- Optional parameters
- Classes

Optional parameters

- In some cases, it may make sense to be able to call a function with a different number of parameters.
    - if we call it with fewer, some of the parameters will take a default value.
    - if we call it with more, we can assign those values.
- We have seen a few examples of this already:
    - `range(10)` vs. `range(1,10)`
    - `l = [1, 2, 3]`
      `l.pop()` vs `l.pop(1)`
- These are called optional parameters.

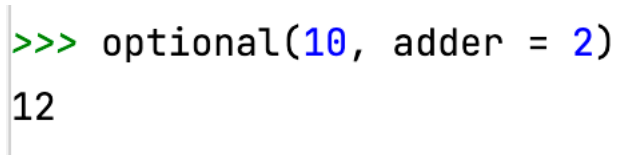
`optional_parameters.py`

- To specify an optional parameter, you declare them like normal parameters, but give them a default value using '='.
- The function `optional` has two optional parameters, so we can call it with 1, 2, or 3 arguments.



```
>>> optional(10)
10
>>> optional(10, 4)
40
>>> optional(10, 4, 7)
47
```

- We can also specify parameters by name.



```
>>> optional(10, adder = 2)
12
```

- Look into the `list_of_nums` function.

Lecture 16: More classes

- Optional parameters
- Classes

queue_structure.py

- Remember, a "class" is the blueprint describing what data and methods an object will have.
- Look at the `Queue` class in `queue_structure.py`
  - It has 5 methods (constructor, `str`, and three other methods)
  - What data does it keep, i.e. what are the instance variables?
    - just `self.queue`, which is a list
  - The constructor has an optional parameter and can be called with either zero parameters or with a list.
    - if it's given a list as a parameter it *copies* it using slicing (:) and saves that away in the instance variable.
    - Why copy it? To avoid aliasing! Otherwise, the instance variables (`self.queue`) would reference the same list as was passed in (a bad thing!)

`queue_structure.py`

- What does this class represent?
- A queue is a data structure (a structure to store data) that is implemented like a line/queue.
    - First things to be added are the first things to be removed.
    - This is known as FIFO (first in first out).
- `add` adds elements to the **end** of the list.
- `remove` removes elements from the **front** of the list.
- `is_empty` just checks if the queue has anything in it.
- Notice that underneath the covers, a queue is just a list. By hiding the list in the class, we have:
    - provided a clear small set of methods that defines how we can interact with the object (the queue).
    - hid the implementation details from whoever uses it.
- We used a list, but could have used something else.
- In a similar way, we could have added to the front of the list and removed from the back and still achieved exactly the same functionality.

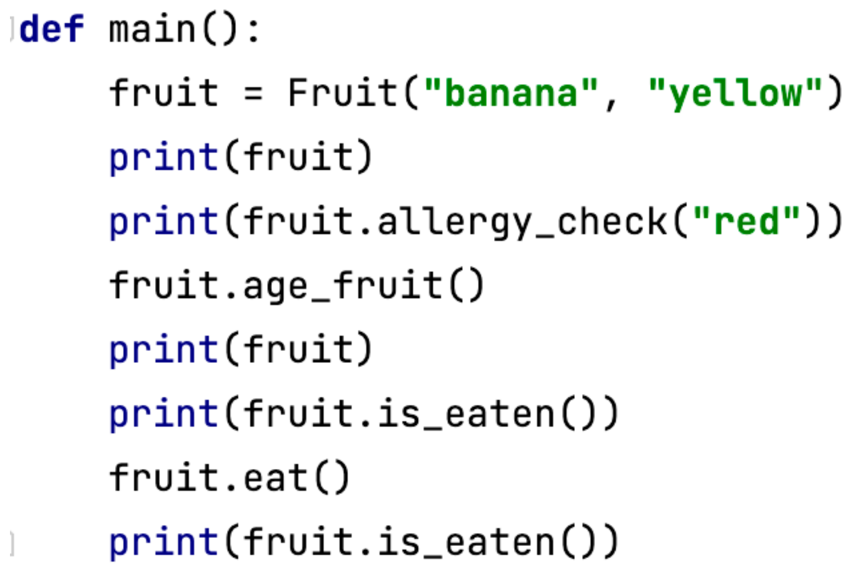
`stack_structure.py`

- What does the `Stack` class represent?
- A stack is a data structure that is implemented like a stack of plates.
  - First things to be added are the last things to be removed.
  - This is known as LIFO (last in first out).
- `add` adds elements to the **top** of the list.
- `remove` removes elements from the **top** of the list.
- `is_empty` just checks if the stack has anything in it.

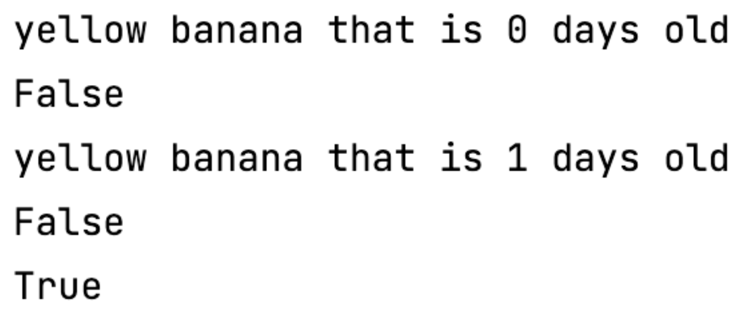
Practice Time

- We're going to design a `Fruit` class. It will have the following constructor and methods:

-
  ```
  def __init__(self, name, color):
          self.name = name
          self.color = color
          self.eaten = False
          self.age = 0
  ```
- `is_eaten` has zero parameters and returns a boolean indicating whether or not the fruit is eaten.
- `eat` has zero parameters and "eats" the fruit.
- `allergy_check` takes a color and returns true if the fruit's color is the same as the input color, false otherwise.
- `age_fruit` takes zero arguments and ages the fruit by a day
- `__str__` prints out a string version of the fruit



```
def main():
    fruit = Fruit("banana", "yellow")
    print(fruit)
    print(fruit.allergy_check("red"))
    fruit.age_fruit()
    print(fruit)
    print(fruit.is_eaten())
    fruit.eat()
    print(fruit.is_eaten())
```



```
yellow banana that is 0 days old
False
yellow banana that is 1 days old
False
True
```

`rectangle3.py`

- A third version of the `Rectangle` class that we saw last week.
- Like the code from `rectangle2.py`, we keep track of the x,y coordinates of the bottom left corner and the width and height
- If we print out the rectangle we see the position of the rectangle and the area.
- In the `__str__` method, we call the `area` method.
- Anytime you want to call another method from within the class you write `self.method_name`, e.g., `self.area()`
- The `equals` method takes one parameter as input: another rectangle!
    - in the body of the method then there are two rectangles: this (`self`) and `another_rectangle`
- We can access the instance variables of the parameter rectangle (`another_rectangle`) in the same way we can access `self`.

Identity

- When you create an object in Python, it has a unique id
  - You can find it using the `id` function which returns a long int.
- Exception: small numbers (between -5 and 256) and some strings that are equal, have the same id.

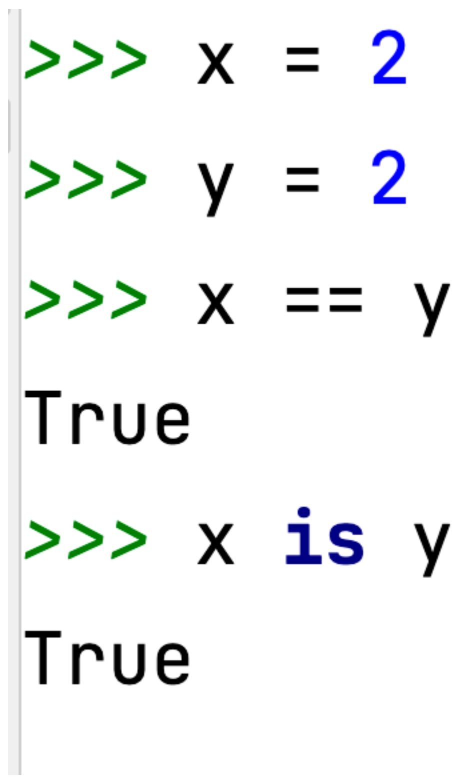
```
>>> list1 = [1, 2, 3]
>>> id(list1)
140178080343104
```

```
>>> x = 2
>>> id(x)
140178605926736
>>> y = 2
>>> id(y)
140178605926736
```

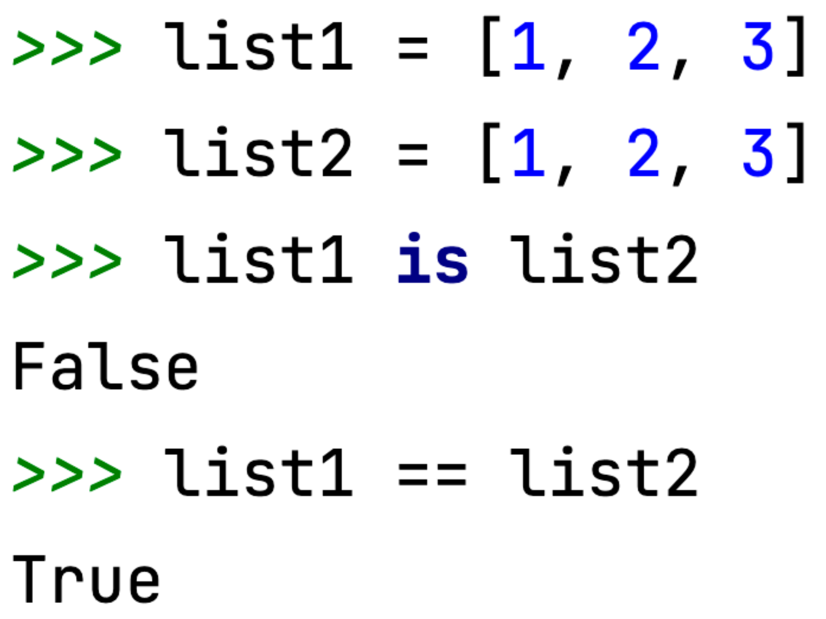
```
>>> list1 = [1, 2, 3]
>>> list2 = [1, 2, 3]
>>> id(list1)
140178080351360
>>> id(list2)
140178080351680
```

Identity vs equality

- When using the `is` operator, Python compares ids.
- When using the `==` operator, Python compares contents of the objects.
- Exception: for small ints and some strings, `is` and `==` will return the same results.



```
>>> x = 2
>>> y = 2
>>> x == y
True
>>> x is y
True
```



```
>>> list1 = [1, 2, 3]
>>> list2 = [1, 2, 3]
>>> list1 is list2
False
>>> list1 == list2
True
```

`__eq__` method

- When creating custom classes, you can implement the `__eq__` method which allows you to compare two objects of your class using the == operator.
- Look at the `__eq__` method in `rectangle3.py` and how it is implicitly used in the `main` function.

Resources

- Textbook: Chapter 17 and Chapter 18
- optional_parameters.py
- queue_structure.py
- stack_structure.py
- fruit.py
- rectangle3.py

Homework

- Assignment 8